

DOCUMENT RESUME

ED 160 083

IR 006 242

AUTHOR Gerhold, George; And Others
TITLE Maxi CAI with a Micro.
PUB DATE Mar 78
NOTE 12p.; Paper presented at the Annual Meeting of the Association for the Development of Computer Based Instructional Systems (Dallas, Texas, March 1-4, 1978); For related document, see IR 006 231

EDRS PRICE MF-\$0.83 Plus Postage. HC Not Available from EDRS.
DESCRIPTORS *Computer Assisted Instruction; Computers; Computer Storage Devices; Cost Effectiveness; *Instructional Systems; *Programming Languages; Systems Development
IDENTIFIERS *Microprocessors

ABSTRACT

This paper describes an effective microprocessor-based CAI system which has been repeatedly tested by a large number of students and edited accordingly. Tasks not suitable for microprocessor based systems (authoring, testing, and debugging) were handled on larger multi-terminal systems. This approach requires that the CAI language used on the microprocessor-based system either be identical to the language used on the authoring system or be a subset of that language. WTS-PILOT-BASIC, described in the appendix, is the current language combination developed and used in the system. The resulting system is available in two versions: one which implements all of WTS PILOT and BASIC with floating point, the four arithmetic operations, two-dimensional numeric arrays, and the standard set of BASIC string operations; and a full version with a complete set of scientific functions, string arrays, and disk-read and disk-write capabilities. (VT)

* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY.

MAXI CAI WITH A MICRO

by

George Gerhold
Larry Kheriaty
Melvin Davidson

"PERMISSION TO REPRODUCE THIS
MATERIAL IN MICROFICHE ONLY
HAS BEEN GRANTED BY

John J. Hirschbuhl

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC) AND
USERS OF THE ERIC SYSTEM."

ED160083

IR006242

Maxi CAI with a Micro

George Gerhold, Larry Kheriaty, Melvin Davidson

Western Washington University

Bellingham, Wash. 98225

The potential impact of microelectronics on human society has been compared in magnitude to the impact of the development of the printing press and to the impact of the industrial revolution. However overblown such comparisons ultimately prove to be, there is little question that a microelectronic revolution is about to occur in CAI. Microprocessors already have been incorporated into sophisticated CAI terminals; microcomputers open the possibility of avoiding the large initial capital investment previously needed to initiate CAI, and microprocessors are making possible TV and arcade games of increasing complexity which are paving the way for public acceptance of CAI. Finally secondary schools which have thus far remained innocent of the computer are purchasing microcomputers for their advanced mathematics programs. As is the case with all technology, microcomputers can be used either effectively or poorly, a truism which is especially applicable to CAI. In this paper we describe an effective microprocessor-based CAI system which we have assembled, and we discuss our reasons for describing the system as effective.

We based the design of our system on two assumptions. First, a microprocessor-based CAI system must be able to deliver CAI of quality comparable to that delivered by the most flexible and powerful minicomputer-based systems; and second, the best CAI courses are those which have been tested by a large number of students and which have been edited repeatedly on the basis of this testing. Our second assumption implies that authoring, testing, and debugging of courses are activities which are not suitable for microprocessor-based systems; these tasks are better handled on larger, multi-terminal systems. Therefore we chose to create a microprocessor-based system for the delivery of CAI courses which will be written and tested on our existing terminal system. This eliminates the need for extensive editing and error-checking resources within the system, which in turn allows us to provide for delivery of top-quality CAI using a 20K byte microcomputer. Thus we have traded complete independence of the microprocessor-based system for maximum sophistication and flexibility of the delivered CAI. It is obvious that it will be relatively easy to add rudimentary authoring and editing capabilities to our system, and we intend to do this in future versions, thus converting it to a completely independent system. However, we do not intend to use these capabilities for writing CAI courses for the reasons discussed.

Our approach requires that the CAI language used on the microprocessor-based system either be identical to the language used on

the authoring system or be a subset of that language. In the latter case the conversion to the subset should be primarily via machine translation. Fortunately our campus system, which we use for authoring, uses two languages for CAI which are well-suited for microprocessor implementation, PILOT and BASIC. WTS-PILOT-BASIC is described in the appendix. It is sufficient to note that we have been able to translate a variety of CAI materials from a variety of sources into this language, without difficulty and usually with improved response time. PILOT, which in its rudimentary, original form was developed for a microprocessor-based terminal, is especially suited to microprocessors in two ways. PILOT uses only a small number of op codes, and it uses a minimum amount of storage space. (It is interesting to note that most CAI languages use a large amount of storage to keep the data necessary to implement particular instructional strategies, and that space and time is used whether or not that strategy is in use.) BASIC - in some dialect - is at present the standard microprocessor computational language. Therefore we chose to stay with our current language combination for the microprocessor-based system.

Our initial thought was that the simplest implementation of a PILOT-BASIC combination could be achieved by addition of the PILOT op codes to an existing BASIC system. This approach was discarded for several reasons. Available microprocessor BASIC operating systems of sufficient power occupy at least 8K bytes of memory; addition of PILOT would require about another 4K. Moreover, these versions of BASIC do

not include all the string functions necessary for implementation of a powerful match instruction. Finally, a crude analysis of existing CAI courses showed that a small minority of the instructions included BASIC syntax. We therefore decided to write an operating system which would handle all the PILOT op codes directly, and then to add the necessary parts of BASIC to this system.

ALL BASIC programs can be translated to a simple form of the language which uses only five instructions: LET, DIM, INPUT, PRINT, and IF — THEN. We chose to write programs on our minicomputer system which would translate existing BASIC programs into the necessary five primitive instructions. (Some instructions require hand translation, e.g. MAT, READ, DATA, OPEN.) Moreover, wherever possible we replace the BASIC instruction with the PILOT equivalent. Thus the BASIC is implemented as follows:

<u>Elementary BASIC</u>	<u>Micro-PILOT</u>
DIM	C: DIM
LET	C: LET
IF () THEN	OP CODE ():
INPUT	A:
PRINT	T:

We should emphasize that the author need not be aware of either Elementary BASIC or Micro PILOT; the author in most cases can program in extended BASIC (and WTS PILOT) and rely on machine translation to make the conversion.

Our two assumptions also influenced our selection of hardware. Obviously we sought to design the lowest cost system which also provided the desired capabilities. The major choices here are bulk storage and terminals. Although very cheap cassette decks have been used for the home microcomputer market, they are not suitable for CAI. Reversible, high-speed, quality decks are still cheaper than floppy disks, but in our opinion the savings is not worth the loss of branching speed and subroutine accessibility which disks provide. We therefore chose to use a 300K byte floppy disk. Only experience will tell if the small 90K byte floppy disks would have been a better choice. The cheapest terminal would have been a modified TV monitor, but these are limited to a 16x64 format. Our courses are written for a 24x80 format, so we chose to use such a terminal through an RS 232 interface. However, there is nothing in the systems software which precludes use with the cheaper terminal. The remaining major choice was the microprocessor. We chose the Motorola 6800 because of the cost of the available 6800 based systems. However, other microprocessors are available and would be equally suitable. In fact implementations of this language on the Z80 are underway, and we plan to implement it on the 8080 if demand warrants it.

The resulting system is available in two versions. The minimum version implements all of WTS PILOT and BASIC with floating point, the four arithmetic operations, two-dimensional numeric arrays, and the standard set of BASIC string operations. This system occupies

less than 9K of memory in 6800 code, so a 16K machine allows sufficient space for arrays and string variables. In the full version we add a complete set of scientific functions, string arrays, and disk-read and disk-write capabilities. This version requires a 20K machine (the standard SWTP 6800 with an additional 16K of memory).

In summary we have assembled a microprocessor-based CAI system of real power and flexibility. This system can be assembled from kits for approximately \$2700 without a terminal. The system will be available during the demonstration periods, and we hope you will take time to test its capabilities.

Appendix: A Description of WTS PILOT

Four instructions comprise the central core of PILOT,

- T: type the following text,
- A: accept an edited answer and optionally assign it as the value of the following string and/or number variable,
- M: match the answer with the following one or more acceptable answers or patterns, and
- J: jump to the designated instruction.

To these are added the statements

- R: remark or comment,
- U: use a subroutine at the designated label,
- E: end the subroutine and return to the place of call, and
- C: calculate using the following BASIC statement.

We supplement these locally with

- : continue the previous T: statement,
- PR: a new problem or course frame,
- XI: execute immediate the following character string as a PILOT statement, and
- SJ: jump to a new segment, preserving symbols but clearing labels.

The values of arithmetic and character string variables can be included in the text of a T: command if preceded by a # or a \$, respectively, (e.g., T: #X3, T: \$B\$). This feature can be used to send special characters to terminals for graphics control. For example, the sequence

C: Let B\$ = CHR(8)

T: \$B\$

will generate a back space on the Ann Arbor terminals. One line of input is accepted during by A: instruction, multiple spaces are compressed to single spaces, automatic upper/lower case editing is performed as specified, and the response optionally may be saved in a string and/or numerical variable. The last response accepted can be compared with a specified pattern string which may include the special characters * to match one character, % to signify space or end, & to match multiple characters, and ! to match one string or another. The result of the match is either an internally kept yes or no.

Any command can be made conditional by adding the following between the command code and the colon following it:

- Y the instruction will be executed only if the last match was successful.
- N the instruction will be executed only if the last match was not successful.
- E the instruction will be executed only if an error flag was turned on.

C the instruction will be executed only if the last relational conditioner was true (see below).

digit 0 through 9 - the instruction will execute only when the last A: instruction has been used repetitively in the designated number of times.

In addition execution of any instruction is made conditional by enclosing a BASIC logical expression (relational conditioner) in parentheses between the command and its following colon. The command will be executed only if the comparison is true, (e.g., T(X<3):).

Any instruction can be labelled by an asterisk followed by from one to six characters, (e.g., *LAB1 T:).

The destination of the J: command is a label (J: label), a line number (J: number), the last A: (e.g., J: @A), the next M: (e.g., J: @M), or the next PR: (e.g., J: @P).

Three modifiers are used to modify the execution of particular op codes. They are

J modifier	MJ: Produces automatic jump to the next M - on a no-match condition.
S modifier	MS: Allows one mismatched character or one pair of characters to be interchanged without triggering a no-match condition.
H modifier	TH: Suppresses the carriage return and line feed at the end of the line.

There are some limitations on the BASIC statements which can be included directly in a C: instruction, but because one of the legal statements is the CALL statement, the C: instruction imbeds the entire extended BASIC language into PILOT.

Programs begin with a header containing the optional automatic features listed:

- U to translate all input to upper case,
- L to translate all input to lower case,
- S to remove all spaces from the input,
- H to enable the HELP command,
- G to enable to GOTO command, and
- W to erase the label table

Additional headers can be imbedded within the course.